<table>
<thead>
<tr>
<th><strong>Title:</strong></th>
<th>Testing MCNP Random Number Generators</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s):</strong></td>
<td>Yasunobu Nagaya &amp; Forrest B. Brown</td>
</tr>
<tr>
<td><strong>Intended for:</strong></td>
<td>MCNP References, historical document from 2002 unpublished memos</td>
</tr>
</tbody>
</table>
Testing MCNP
Random Number Generators

Forrest B. Brown
Diagnostics Applications Group (X-5)
Los Alamos National Laboratory
<fbrown@lanl.gov>

Yasunobu Nagaya
Visiting Scientist - LANL
Japan Atomic Energy Research Institute
<nagaya@lanl.gov>
Testing MCNP Random Number Generators

- **Introduction**
  - History
  - Requirements

- **MCNP-5 RN Generator**
  - Algorithm
  - Coding
  - Skip-ahead
  - Parallel considerations

- **RN Generator Parameters**
  - Traditional generators – MCNP, RACER, RCP, MORSE, KENO, VIM, EGS
  - Extended generators – 63-bits

- **RN Generator Testing**
  - Knuth statistical tests
  - Marsaglia’s DIEHARD test suite
  - Spectral test
  - Results

- **Future Plans**
Introduction
Introduction

- **Monte Carlo Simulation:**
  - Random sampling to model the outcome of physical events
  - Ray tracing through 3-D computational geometry

- **Random Number Generators**
  - *Numbers* are not random; a *sequence* of numbers can be
  - Repeatable (deterministic)
  - Pass statistical tests for randomness
  
  - Function which generates a sequence of numbers which appear to have been randomly sampled from a uniform distribution on \((0,1)\)
  
  - Probability density function
    
    \[
    \begin{array}{c}
    \text{0} \\
    \hline
    \text{1} \\
    \end{array}
    \]

  - Typical use in codes: \(r = \text{rang}()\)
Remarks: History

- **MCNP & related precursor codes**
  - 40+ years of intense use
  - Many different computers & compilers
  - Modern versions are parallel: MPI + threads
  - History based: Consecutive RNs used for primary particle, then for each of it’s secondaries in turn, etc.
  - RN generator is small fraction of total computing time (~ 5%)

- **Traditional MCNP RN Algorithm**
  - Linear congruential, multiplicative
    \[ S_{n+1} = g \cdot S_n \mod 2^{48}, \quad g = 5^{19} \]
  - 48-bit integer arithmetic, carried out in 24-bit pieces
  - Stride for new histories: 152,917
  - Skip-ahead: crude, brute-force
  - Period / stride = 460 x 10^6 histories
  - Similar RN generators in RACER, RCP, MORSE, KENO, VIM
Remarks:  Requirements

- **Algorithm**
  - Robust, well-proven
  - Long period: $10^9$ particles $\times$ stride 152,917 $= 10^{14}$ RNs
  - $>10^9$ parallel streams
  - High-precision is **not** needed, low-order bits not important
  - Reasonable theoretical basis, no correlation within or between histories

- **Coding**
  - Robust !!!! Must never fail.
  - Rapid initialization for each history
  - Minimal amount of state information
  - Fast, but portable – must be exactly reproducible on any computer/compiler
MCNP-5

RN Generator
**MCNP-5 RN Generator: Algorithm**

- **Linear congruential generator (LCG)**

\[ S_{n+1} = g \times S_n + c \mod 2^m, \]

**Period** = \( 2^m \) (for \( c > 0 \)) or \( 2^{m-2} \) (for \( c = 0 \))

- **Traditional MCNP:** \( m=48, \ c=0 \) \( \text{Period}=10^{14}, \text{48-bit integers} \)
- **MCNP-5:** \( m=63, \ c=1 \) \( \text{Period}=10^{19}, \text{63-bit integers} \)

**How to pick \( g \) and \( c \)??**

- **RN Sequence & Particle Histories**

  ![Sequence Example](124x123)

- **Stride for new history:** \( 152,917 \)
MCNP-5 RN Generator: Skip-ahead

• To skip ahead \( k \) steps in the RN sequence:

\[
S_k = g \cdot S_{k-1} + c \mod 2^m
\]
\[
= g^k \cdot S_0 + c \left(\frac{g^{k-1}}{g-1}\right) \mod 2^m
\]

• Negative skip \( k \) equivalent to positive skip \([\text{period-}k]\)

• Can skip from any seed to any other
  – Initial seed \( \rightarrow \) \( i^{\text{th}} \) seed for \( j^{\text{th}} \) particle on \( m^{\text{th}} \) processor in \( k^{\text{th}} \) generation
  – Particle \( i \rightarrow \) particle \( j \)
  – Batch \( i \rightarrow \) batch \( j \)

• Need a fast way to compute \( g^k \mod 2^m \) & \( c\left(\frac{g^{k-1}}{g-1}\right) \mod 2^m \) in \( O(m) \) steps, rather than \( O(k) \) steps
**MCNP-5 RN Generator:** Skip-ahead

- **Computing** \( G = g^k \mod 2^m \)

  \[
  G \leftarrow 1, \quad h \leftarrow g, \quad i \leftarrow k+2^m \mod 2^m
  \]

  While \( i > 0 \)
  
  if \( i = \text{odd} \)
  \[
  G \leftarrow G \cdot h \mod 2^m
  
  h \leftarrow h^2 \mod 2^m
  
  i \leftarrow \lfloor i / 2 \rfloor
  \]

  Used in: RACER, VIM, KENO-Va (Spain), MCNP-5

- **Computing** \( C = c(g^{k-1})/(g-1) \mod 2^m \)

  \[
  C \leftarrow 0, \quad f \leftarrow c, \quad h \leftarrow g, \quad i \leftarrow k+2^m \mod 2^m
  \]

  While \( i > 0 \)
  
  if \( i = \text{odd} \)
  \[
  C \leftarrow C \cdot h + f \mod 2^m
  
  f \leftarrow f \cdot (h+1) \mod 2^m
  
  h \leftarrow h^2 \mod 2^m
  
  i \leftarrow \lfloor i / 2 \rfloor
  \]

MCNP-5 RN Generator: Coding

- RN Generation in MCNP-5
  - RN module, entirely replaces all previous coding for RN generation
  - Fortran-90, using INTEGER(I8) internally, where I8=selected_int_kind(18)
  - All parameters, variables, & RN generator state are PRIVATE, accessible only via “accessor” routines
  - Includes “new” skip-ahead algorithm for fast initialization of histories, greatly simplifies RN generation for parallel calculations
  - Portable, standard, thread-safe
  - Built-in unit test, compile check, and run-time test
  - Developed on PC, tested on SGI, IBM, Sun, Compaq
MCNP5 RN Generator: Coding

Module mcnp_random

```plaintext
  integer(I8), PRIVATE, SAVE :: RN_MULT, ! Multiplier
  & RN_ADD,  ! Adder
  & RN_MASK, ! Mask, to get lower bits

  real(R8), PRIVATE, SAVE :: RN_NORM ! norm to (0,1)

! Private data for a single history
!
  integer(I8), PRIVATE :: RN_SEED, RN_COUNT
  !$OMP THREADPRIVATE ( /RN_THREAD/ )

CONTAINS

  function rang( )
  ! MCNP5 random number generator
  implicit none
  real(R8) :: rang

  RN_SEED  = iand( RN_MULT*RN_SEED, RN_MASK )
  RN_SEED  = iand( RN_SEED+RN_ADD, RN_MASK )
  rang     = RN_SEED * RN_NORM
  RN_COUNT = RN_COUNT + 1
  return
end function rang
```

!$OMP END PARALLEL
Program mcnp5

! Initialize RN parameters for problem
call RN_init_problem( new_standard_gen= 2, &
                  & new_seed= ProblemSeed )


do nps = 1, number_of_histories

  ! Analyze one particle history
  call RN_init_particle( nps )

  if( rang() > xs ) . . .

  ! Terminate history
  call RN_update_stats
MCNP-5 Random Number Generation & Testing

• **Introduction**
  ✓ History
  ✓ Requirements

• **MCNP-5 RN Generator**
  ✓ Algorithm
  ✓ Coding
  ✓ Skip-ahead
  ✓ Parallel considerations

• **RN Generator Parameters**
  – Extended generators – 63-bits
  – L’Ecuyer’s 63-bit generators

• **RN Generator Testing**
  – Knuth statistical tests
  – Marsaglia’s DIEHARD test suite
  – Spectral test
  – Performance test
  – Results

• **Future Plans**
Extended generators : 63-bit LCGs

- Selection of multiplier, increment and modulus

\[ S_{n+1} = 5^{19} S_n + 0 \mod 2^{48} \text{ (MCNP4)} \]

\[ 5^{23}, 5^{25}, 1, 2^{63} \]

- Multiplicative LCG\((g, 0, 2^\beta)\)

\[ g \equiv \pm 3 \mod 8, \ S_0 = \text{odd} \quad \Rightarrow \quad \text{Period} : 2^{\beta-2} \]

- Mixed LCG\((g, c, 2^\beta)\)

\[ g \equiv 1 \mod 4, \ c = \text{odd} \quad \Rightarrow \quad \text{Period} : 2^\beta \]

- Extension of multiplier
  - \(5^{19} = 45\)-bit integer in the binary representation
  - \(5^{19}\) seems to be slightly small in 63-bit environment.
  - Odd powers of 5 satisfy both conditions above.
L’Ecuyer’s 63-bit LCGs

- Good multipliers are chosen based on the spectral test.
- Multiplicative LCGs
  - LCG(3512401965023503517, 0, 2^{63})
  - LCG(2444805353187672469, 0, 2^{63})
  - LCG(1987591058829310733, 0, 2^{63})
- Mixed LCGs
  - LCG(92197414264999971445, 1, 2^{63})
  - LCG(2806196910506780709, 1, 2^{63})
  - LCG(3249286849523012805, 1, 2^{63})
- Tested RNGs
  - Traditional MCNP RNG
  - 6 Extended 63-bit LCGs
  - L’Ecuyler’s 63-bit LCGs above
  - 13 LCGs were tested.
RN Generator Testing
Tests for RNGs

• **Theoretical tests**:  
  – Analyzing the algorithm of RNGs based on the number theory and the theory of statistics.  
  – Theoretical tests depend on the type of RNGs. (LCG, Shift register, Lagged Fibonacci, etc.)  
  – LCG: **Spectral test**

• **Empirical tests**:  
  – Analyzing the uniformity, patterns, etc. of RNs generated by RNGs.  
  – **Standard tests** (reviewed by D. Knuth): SPRNG test routines  
  – Bit level tests (**DIEHARD test** proposed by G. Marsaglia): more stringent  
  – Physical tests: RNGs are used in a practical application. The exact solutions for the tests are known. (not performed in this work)
Standard test suite in SPRNG

- **SPRNG (Scalable Parallel Random Number Generators)**
  - Test programs are available. http://sprng.cs.fsu.edu

- **Standard test suite**
  - Equidistribution, Serial, Gap, Poker, Coupon collector’s, Permutation, Runs-up, Maximum-of-t, Collision tests

- **Choice of test parameters**
  - Mascagni’s test suite: Submitted to Parallel Computing
Equidistribution test

- Check whether RNs are uniformly generated in [0, 1).
- Generate random integers in [0,d-1].
- Each integer must have the equal probability 1/d.

0.10574, 0.66509, 0.46622, 0.93925, 0.26551, 0.11361, …

$\left\lfloor d \times \xi_i \right\rfloor$

0, 5, 3, 7, 2, 0, 2, 3, 1, 4, …

Count frequencies of 0 ~ d-1.

Cumulative chi-square distribution

Cumulative chi-square distribution

$V = \sum_{s=1}^{k} \frac{(Y_s - np_s)^2}{np_s}$
Criterion of “Pass or Failure”

- All empirical tests score a statistic.
- A goodness-of-fit test is performed on the test statistic and yield a p-value. (Chi-square or Kolmogorov-Smirnov test)
- If the p-value is close to 0 or 1, a RNG is suspected to fail.
- Significance level : 0.01(1%)
- Repeat each test 3 times.
- All 3 p-values are suspicious, then the RNG fails.
DIEHARD test suite

**DIEHARD test**
- A battery of tests proposed by G. Marsaglia.
- Test all bits of random integers, not only the most significant bits.
- More stringent than standard tests.
- Test programs are available. http://stat.fsu.edu/~geo/diehard.html

**Included tests**
- Birthday spacings, Overlapping 5-permutation, Binary rank, Bitstream, Overlapping-pairs-sparse-occupancy (OPSO), Overlapping-quadruples-sparse-occupancy (OQSO), DNA, Count-the-1's test on a stream of bytes, Count-the-1's test for specific bytes, Parking lot, Minimum distance, 3-D spheres, Squeeze, Overlapping sums, Runs, Craps

**Test Parameters**
- Default test parameters were used in this work.
Overlapping-pairs-sparse-occupancy test (1)

- **OPSO** = Overlapping-Pairs-Sparse-Occupancy test
- **Preparation of 32-bit integers**
  
  \[ 2^{32} \xi_i \]
  
  0.10574, 0.66509, 0.46622, 0.93925, 0.26551, 0.11361, ...
  
  454158374, 2856527213, 2002411287, 4034027575, ...

- **Binary representation**
  
  11011000110010001111010000001100110,
  
  101010101000110010010101101101,
  
- **Letter**: a designated string of consecutive 10 bits
  
  11011000100011110111010000100110,
  
  10101010100001100100101011011010110101101,
  
  **Letter**: \(2^{10} = 1024\) patterns
  
  (letters)
Overlapping-pairs-sparse-occupancy test (2)

- 2-letter words are formed from an alphabet of 1024 letters.

0000100110, 0101101101, 1100010111, 0000110111, ...

Decimal representation
38, 365, 791, 55, ...

- Count the number of missing words (=j).
- The number of missing words should be very closely normally distributed with mean 141,909, standard deviation 290.

\[ z = \frac{j - 141909}{290} \]
Overlapping-quadruples-sparse-occupancy test

- OQSO = Overlapping-Quadraples-Sparse-Occupancy test
- Similar to the OPSO test.
- Letter: a designated string of consecutive 5 bits
  \[11011000100011110100000100110, 10101010010000110010010101101101, \ldots\]
- 4-letter words are formed from an alphabet of 32 letters.
  \[00110, 01101, 10111, 10111, \ldots\]
- The number of missing words should be very closely normally distributed with mean 141909, standard deviation 295.
DNA test

• Similar to the OPSO and OQSO tests.

- **Letter**: a designated string of consecutive 2 bits
  
  \[11011000100011110100000100110,\]
  \[10101001000011001001011101101,\] …

- Letter : \(2^2 = 4\) letters

• **10-letter words** are formed from an alphabet of 4 letters.

  \[10, 1, 11, 11, 11, 1, 10, 0, 11, 10, \] …

  10-letter word

• The number of missing words should be very closely normally distributed with mean 141909, standard deviation 399.
Criterion for DIEHARD test

- If the p-value is close to 0 or 1, a RNG is suspected to fail.

- Significance level : 0.01(1%)

- A RNG fails the test if we get six or more p-values less than 0.01 or more than 0.99.
Results for standard & DIEHARD tests

- All 13 RNGs pass all standard tests with L’Ecuyer’s, Vattulainen’s and Mascagni’s test parameters.

- Extended and L’Ecuyer’s 63-bit LCGs pass all the DIEHARD tests.

- The traditional MCNP RNG fails the OPSO, OQSO and DNA tests in the DIEHARD test suite.
### Result of OPSO test for traditional MCNP RNG

<table>
<thead>
<tr>
<th>Tested bits</th>
<th>p-value</th>
<th>Tested bits</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits 23 to 32</td>
<td>0.0000</td>
<td>bits 11 to 20</td>
<td>0.7457</td>
</tr>
<tr>
<td>bits 22 to 31</td>
<td>0.0000</td>
<td>bits 10 to 19</td>
<td>0.0598</td>
</tr>
<tr>
<td>bits 21 to 30</td>
<td>0.0000</td>
<td>bits 9 to 18</td>
<td>0.1122</td>
</tr>
<tr>
<td>bits 20 to 29</td>
<td>0.0000</td>
<td>bits 8 to 17</td>
<td>0.4597</td>
</tr>
<tr>
<td>bits 19 to 28</td>
<td>0.0001</td>
<td>bits 7 to 16</td>
<td>0.0011</td>
</tr>
<tr>
<td>bits 18 to 27</td>
<td>0.6639</td>
<td>bits 6 to 15</td>
<td>0.6319</td>
</tr>
<tr>
<td>bits 17 to 26</td>
<td>0.0445</td>
<td>bits 5 to 14</td>
<td>0.7490</td>
</tr>
<tr>
<td>bits 16 to 25</td>
<td>0.0125</td>
<td>bits 4 to 13</td>
<td>0.2914</td>
</tr>
<tr>
<td>bits 15 to 24</td>
<td>0.7683</td>
<td>bits 3 to 12</td>
<td>0.1792</td>
</tr>
<tr>
<td>bits 14 to 23</td>
<td>0.9712</td>
<td>bits 2 to 11</td>
<td>0.3253</td>
</tr>
<tr>
<td>bits 13 to 22</td>
<td>0.1077</td>
<td>bits 1 to 10</td>
<td>0.7277</td>
</tr>
<tr>
<td>bits 12 to 21</td>
<td>0.0717</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Result of OQSO test for traditional MCNP RNG

<table>
<thead>
<tr>
<th>Tested bits</th>
<th>p-value</th>
<th>Tested bits</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits 28 to 32</td>
<td>1.0000</td>
<td>bits 14 to 18</td>
<td>0.6487</td>
</tr>
<tr>
<td>bits 27 to 31</td>
<td>1.0000</td>
<td>bits 13 to 17</td>
<td>0.5575</td>
</tr>
<tr>
<td>bits 26 to 30</td>
<td>1.0000</td>
<td>bits 12 to 16</td>
<td>0.1634</td>
</tr>
<tr>
<td>bits 25 to 29</td>
<td>1.0000</td>
<td>bits 11 to 15</td>
<td>0.6600</td>
</tr>
<tr>
<td>bits 24 to 28</td>
<td>1.0000</td>
<td>bits 10 to 14</td>
<td>0.2096</td>
</tr>
<tr>
<td>bits 23 to 27</td>
<td>1.0000</td>
<td>bits 9 to 13</td>
<td>0.3759</td>
</tr>
<tr>
<td>bits 22 to 26</td>
<td>0.0000</td>
<td>bits 8 to 12</td>
<td>0.9191</td>
</tr>
<tr>
<td>bits 21 to 25</td>
<td>0.0000</td>
<td>bits 7 to 11</td>
<td>0.8554</td>
</tr>
<tr>
<td>bits 20 to 24</td>
<td>0.0000</td>
<td>bits 6 to 10</td>
<td>0.5535</td>
</tr>
<tr>
<td>bits 19 to 23</td>
<td>0.1906</td>
<td>bits 5 to 9</td>
<td>0.4955</td>
</tr>
<tr>
<td>bits 18 to 22</td>
<td>0.0011</td>
<td>bits 4 to 8</td>
<td>0.0868</td>
</tr>
<tr>
<td>bits 17 to 21</td>
<td>0.3823</td>
<td>bits 3 to 7</td>
<td>0.1943</td>
</tr>
<tr>
<td>bits 16 to 20</td>
<td>0.8394</td>
<td>bits 2 to 6</td>
<td>0.8554</td>
</tr>
<tr>
<td>bits 15 to 19</td>
<td>0.2518</td>
<td>bits 1 to 5</td>
<td>0.7421</td>
</tr>
</tbody>
</table>
### Result of DNA test for traditional MCNP RNG

<table>
<thead>
<tr>
<th>Tested bits</th>
<th>p-value</th>
<th>Tested bits</th>
<th>p-value</th>
<th>Tested bits</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits 31 to 32</td>
<td>1.0000</td>
<td>bits 20 to 21</td>
<td>0.4937</td>
<td>bits 9 to 10</td>
<td>0.4550</td>
</tr>
<tr>
<td>bits 30 to 31</td>
<td>1.0000</td>
<td>bits 19 to 20</td>
<td>0.0613</td>
<td>bits 8 to 9</td>
<td>0.4737</td>
</tr>
<tr>
<td>bits 29 to 30</td>
<td>1.0000</td>
<td>bits 18 to 19</td>
<td>0.2383</td>
<td>bits 7 to 8</td>
<td>0.7834</td>
</tr>
<tr>
<td>bits 28 to 29</td>
<td>1.0000</td>
<td>bits 17 to 18</td>
<td>0.4831</td>
<td>bits 6 to 7</td>
<td>0.4063</td>
</tr>
<tr>
<td>bits 27 to 28</td>
<td>1.0000</td>
<td>bits 16 to 17</td>
<td>0.0925</td>
<td>bits 5 to 6</td>
<td>0.8959</td>
</tr>
<tr>
<td>bits 26 to 27</td>
<td>0.1777</td>
<td>bits 15 to 16</td>
<td>0.0197</td>
<td>bits 4 to 5</td>
<td>0.3438</td>
</tr>
<tr>
<td>bits 25 to 26</td>
<td>0.0000</td>
<td>bits 14 to 15</td>
<td>0.7377</td>
<td>bits 3 to 4</td>
<td>0.3972</td>
</tr>
<tr>
<td>bits 24 to 25</td>
<td>0.0000</td>
<td>bits 13 to 14</td>
<td>0.7171</td>
<td>bits 2 to 3</td>
<td>0.8986</td>
</tr>
<tr>
<td>bits 23 to 24</td>
<td>0.0000</td>
<td>bits 12 to 13</td>
<td>0.0309</td>
<td>bits 1 to 2</td>
<td>0.5407</td>
</tr>
<tr>
<td>bits 22 to 23</td>
<td>0.0000</td>
<td>bits 11 to 12</td>
<td>0.2803</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bits 21 to 22</td>
<td>0.0000</td>
<td>bits 10 to 11</td>
<td>0.8440</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tested bits: 31 to 32, 20 to 21, 9 to 10, 8 to 9, 7 to 8, 6 to 7, 5 to 6, 4 to 5, 3 to 4, 2 to 3, 1 to 2, 10 to 11

p-values: 1.0000, 0.4937, 0.4550, 0.4737, 0.7834, 0.4063, 0.8959, 0.3438, 0.3972, 0.8986, 0.5407, 0.8440
Comments on results for OPSO, OQSO, DNA

• Less significant (lower) bits of RNs fail the tests.

• These failures in less significant bits are caused by the shorter period than the significant bits.

Drawback of LCGs with power-of-two moduli

The \((r+1)\)-th most significant bit has period length at most \(2^{-r}\) times that of the most significant bit.

• However, these failures do not have a significant impact in the practical use.
Spectral test

- LCGs have regular patterns (lattice structures) when overlapping $t$-tuples of a random number sequence are plotted in a hypercube. (Marsaglia, 1968).

- All the $t$-tuples are covered with families of parallel $(t-1)$-dimensional hyperplanes.

- The spectral test determines the maximum distance between adjacent parallel hyperplanes.
Illustration of the spectral test

- Example: $S_{n+1} = 137 S_n + 187 \mod 256$

\[
\begin{align*}
0.26562, 0.12109, 0.32031, 0.61328, 0.75000, & \\
\end{align*}
\]

\[
\begin{align*}
\xi_i & \\
\xi_{i+1} & \\
\end{align*}
\]
Measures for criterion & ranking

• **μ value proposed by Knuth**
  – Represent the effectiveness of a multiplier.

  Knuth’s criterion

<table>
<thead>
<tr>
<th>μᵢ(m, g) for 2 ≤ t ≤ 6</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>μᵢ(m, g) ≥ 1</td>
<td>Pass with flying colors</td>
</tr>
<tr>
<td>0.1 ≤ μᵢ(m, g) &lt; 1</td>
<td>Pass</td>
</tr>
<tr>
<td>μᵢ(m, g) ≤ 0.1</td>
<td>Fail</td>
</tr>
</tbody>
</table>

• **S value**
  – Normalized maximum distance.

  \[ S_t = \frac{d_t^*(m)}{d_t(m, g)} \]

  \( d_t^*(m) \): Maximum distance between adjacent parallel hyperplanes.

  \( d_t(m, g) \): Lower bound on \( d_t(m, g) \).

  – The closer to 1 the S value is, the better the RNG is.
Results of spectral test

• Results for the traditional MCNP RNG

<table>
<thead>
<tr>
<th>Dimension(t)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu_t(m,g) )</td>
<td>3.0233</td>
<td>0.1970</td>
<td>1.8870</td>
<td>0.9483</td>
<td>1.8597</td>
<td>0.8802</td>
<td>1.2931</td>
</tr>
<tr>
<td>( S_t(m,g) )</td>
<td>0.9129</td>
<td>0.3216</td>
<td>0.6613</td>
<td>0.5765</td>
<td>0.6535</td>
<td>0.5844</td>
<td>0.6129</td>
</tr>
</tbody>
</table>

• All extended 63-bit LCGs fail with Knuth’s criterion.
• All L’Ecuyer’s 63-bit LCGs pass with flying colors.
• Comparison of minimum S values

<table>
<thead>
<tr>
<th>RNG</th>
<th>Minimum ( S_t(m,g) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCG(5^{19},0,2^{48})</td>
<td>0.3216</td>
</tr>
<tr>
<td>LCG(3512401965023503517,0,2^{63})</td>
<td>0.7493</td>
</tr>
<tr>
<td>LCG(2444805353187672469,0,2^{63})</td>
<td>0.7094</td>
</tr>
<tr>
<td>LCG(1987591058829310733,0,2^{63})</td>
<td>0.6449</td>
</tr>
<tr>
<td>LCG(9219741426499971445,1,2^{63})</td>
<td>0.7371</td>
</tr>
<tr>
<td>LCG(2806196910506780709,1,2^{63})</td>
<td>0.6967</td>
</tr>
<tr>
<td>LCG(3249286849523012805,1,2^{63})</td>
<td>0.6451</td>
</tr>
</tbody>
</table>
Performance test

• Test program

```fortran
integer(8) :: i
integer(8), parameter :: NumGeneratedRNs = 1000000000
real(8)    :: rang ! For MCNP4
real(8)    :: RN_initial, RN_last
double :: dummy

!call random! For MCNP4
calls RN_init_problem(new_standard_gen = 1)
RN_initial = rang()
do i = 2, NumGeneratedRNs - 1
dummy = rang()
end do
RN_last = rang()
```

# Results of Performance Test

- **Comparison between MCNP-4 and -5**
- **Generate 1 billion RNs.**

## Results

<table>
<thead>
<tr>
<th></th>
<th>MCNP4</th>
<th>MCNP5</th>
<th>MCNP4/MCNP5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU (sec)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No optimization</td>
<td>290.0</td>
<td>97.1</td>
<td>3.0</td>
</tr>
<tr>
<td>Local optimization</td>
<td>191.7</td>
<td>77.2</td>
<td>2.5</td>
</tr>
<tr>
<td>Full optimization</td>
<td>188.4</td>
<td>78.1</td>
<td>2.4</td>
</tr>
</tbody>
</table>

---

Platform: Windows 2000, Intel Pentium III 1GHz
Compiler: Compaq Visual Fortran Ver.6.6
Summary

• The traditional MCNP RNG fails the OPSO, OQSO and DNA tests in the DIEHARD test suite.

• The 63-bit LCGs extended from the MCNP RNG fail the spectral test.

• L'Ecuyer's 63-bit LCGs pass all the tests and their multipliers are excellent judging from the spectral test.

• These 63-bit LCGs are implemented in the RNG package for MCNP Ver.5.

• The MCNP-5 RNG is ~2.5 times faster than the MCNP-4 RNG.
Future Work
Plans for MCNP RN Generation

• For now, stick with existing RN algorithm - LCG
  – Today's longest problems use ~10^9 histories, for a total of ~10^{14} RN’s
  – The period of the RN generator in MCNP5 has been extended by a factor of 10^5 from 2^{46} = 7 x 10^{13} to 2^{63} = 9.2 x 10^{18}

• Eventually, will need an even longer period.
  – ASCI: 30 T computer this year, 100 T in a few years, & then ....
  – More histories + RN streams by particle type \rightarrow need longer period

• Desirable to modify MCNP5 so that separate particle types (neutrons, photons, electrons, ...) have separate RN streams
  – Want particle behavior to be identical & reproducible if physics options involving other particle types are turned on/off
  – For example, neutron behavior for collisions, tracking, tallies, etc., should be the same if a problem is run with
    • Neutrons only
    • Neutrons + photons
    • Neutrons + photons + electrons
Plans for MCNP RN Generation

• For independent particle streams, could use a different RN additive constant for each particle type:

  Neutrons: \[ S_{N,n+1} = g S_{N,n} + c_N \mod 2^m \]
  Photons: \[ S_{P,n+1} = g S_{P,n} + c_P \mod 2^m \]
  Electrons: \[ S_{E,n+1} = g S_{E,n} + c_E \mod 2^m \]

  Percus & Kalos have proven that the streams would be independent.

• For a longer period:
  – Could extend RN generator to use more than 64-bits
    • Straightforward coding extensions to existing generator
    • Retain “tried & true” mixed LCG scheme
    • Need new multiplier, adder, modulus, & extensive testing

  – Could use different RN algorithm with longer period
    • Combined LCG’s seems a good bet
    • Retain existing coding & algorithm, combine 2 LCG’s
    • Needs a lot of thought, plus advice from experts
End
Appendices
### Spectral test for extended Multiplicative LCGs

<table>
<thead>
<tr>
<th>Dimension(t)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCG(5^{19},0,2^{63})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_t(m,g)$</td>
<td>1.7321</td>
<td>2.1068</td>
<td>2.7781</td>
<td>1.4379</td>
<td>0.0825</td>
<td>2.0043</td>
<td>5.9276</td>
</tr>
<tr>
<td>$S_t(m,g)$</td>
<td>0.6910</td>
<td>0.7085</td>
<td>0.7284</td>
<td>0.6266</td>
<td>0.3888</td>
<td>0.6573</td>
<td>0.7414</td>
</tr>
<tr>
<td>LCG(5^{23},0,2^{63})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_t(m,g)$</td>
<td>0.0028</td>
<td>1.9145</td>
<td>2.4655</td>
<td>5.4858</td>
<td>0.3327</td>
<td>0.2895</td>
<td>6.6286</td>
</tr>
<tr>
<td>$S_t(m,g)$</td>
<td>0.0280</td>
<td>0.6863</td>
<td>0.7070</td>
<td>0.8190</td>
<td>0.4906</td>
<td>0.4986</td>
<td>0.7518</td>
</tr>
<tr>
<td>LCG(5^{25},0,2^{63})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_t(m,g)$</td>
<td>0.3206</td>
<td>1.8083</td>
<td>0.0450</td>
<td>3.0128</td>
<td>0.3270</td>
<td>3.1053</td>
<td>0.4400</td>
</tr>
<tr>
<td>$S_t(m,g)$</td>
<td>0.2973</td>
<td>0.6733</td>
<td>0.2598</td>
<td>0.7265</td>
<td>0.4892</td>
<td>0.6998</td>
<td>0.5356</td>
</tr>
</tbody>
</table>
## Spectral test for extended Mixed LCGs

<table>
<thead>
<tr>
<th>Dimension(t)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCG(5^{19},1,2^{63})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_t(m,g) )</td>
<td>1.7321</td>
<td>2.9253</td>
<td>2.4193</td>
<td>0.3595</td>
<td>0.0206</td>
<td>0.5011</td>
<td>1.6439</td>
</tr>
<tr>
<td>( S_t(m,g) )</td>
<td>0.6910</td>
<td>0.7904</td>
<td>0.7036</td>
<td>0.4749</td>
<td>0.3086</td>
<td>0.5392</td>
<td>0.6316</td>
</tr>
<tr>
<td>LCG(5^{23},1,2^{63})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_t(m,g) )</td>
<td>0.0007</td>
<td>2.8511</td>
<td>2.5256</td>
<td>3.1271</td>
<td>4.5931</td>
<td>1.8131</td>
<td>4.2919</td>
</tr>
<tr>
<td>( S_t(m,g) )</td>
<td>0.0140</td>
<td>0.7837</td>
<td>0.7112</td>
<td>0.7319</td>
<td>0.7598</td>
<td>0.6480</td>
<td>0.7121</td>
</tr>
<tr>
<td>LCG(5^{25},1,2^{63})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_t(m,g) )</td>
<td>0.0801</td>
<td>3.4624</td>
<td>1.3077</td>
<td>1.0853</td>
<td>1.4452</td>
<td>0.7763</td>
<td>1.3524</td>
</tr>
<tr>
<td>( S_t(m,g) )</td>
<td>0.1486</td>
<td>0.8361</td>
<td>0.6033</td>
<td>0.5923</td>
<td>0.6266</td>
<td>0.5740</td>
<td>0.6163</td>
</tr>
</tbody>
</table>
## Spectral test for L’Ecuyer’s Multiplicative LCGs

<table>
<thead>
<tr>
<th>Dimension(t)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCG(3512401965023503517,0,2^{63})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_t(m,g)$</td>
<td>2.9062</td>
<td>2.9016</td>
<td>3.1105</td>
<td>4.0325</td>
<td>5.3992</td>
<td>6.7498</td>
<td>7.2874</td>
</tr>
<tr>
<td>$S_t(m,g)$</td>
<td>0.8951</td>
<td>0.7883</td>
<td>0.7493</td>
<td>0.7701</td>
<td>0.7806</td>
<td>0.7818</td>
<td>0.7608</td>
</tr>
<tr>
<td>LCG(2444805353187672469,0,2^{63})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_t(m,g)$</td>
<td>2.2588</td>
<td>2.4430</td>
<td>6.4021</td>
<td>2.9364</td>
<td>3.0414</td>
<td>5.4274</td>
<td>4.6180</td>
</tr>
<tr>
<td>$S_t(m,g)$</td>
<td>0.7891</td>
<td>0.7443</td>
<td>0.8974</td>
<td>0.7228</td>
<td>0.7094</td>
<td>0.7579</td>
<td>0.7186</td>
</tr>
<tr>
<td>LCG(1987591058829310733,0,2^{63})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_t(m,g)$</td>
<td>2.4898</td>
<td>3.4724</td>
<td>1.7071</td>
<td>2.5687</td>
<td>2.1243</td>
<td>2.0222</td>
<td>4.1014</td>
</tr>
<tr>
<td>$S_t(m,g)$</td>
<td>0.8285</td>
<td>0.8369</td>
<td>0.6449</td>
<td>0.7037</td>
<td>0.6682</td>
<td>0.6582</td>
<td>0.7080</td>
</tr>
</tbody>
</table>
## Spectral test for L’Ecuyer’s Mixed LCGs

<table>
<thead>
<tr>
<th>Dimension(t)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCG(9219741426499971445,1,2(^{63}))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\mu_t(m,g))</td>
<td>2.8509</td>
<td>2.8046</td>
<td>3.5726</td>
<td>3.8380</td>
<td>3.8295</td>
<td>6.4241</td>
<td>6.8114</td>
</tr>
<tr>
<td>(S_t(m,g))</td>
<td>0.8865</td>
<td>0.7794</td>
<td>0.7757</td>
<td>0.7625</td>
<td>0.7371</td>
<td>0.7763</td>
<td>0.7544</td>
</tr>
<tr>
<td>LCG(2806196910506780709,1,2(^{63}))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\mu_t(m,g))</td>
<td>1.9599</td>
<td>4.0204</td>
<td>4.4591</td>
<td>3.1152</td>
<td>3.0728</td>
<td>3.0111</td>
<td>3.7947</td>
</tr>
<tr>
<td>(S_t(m,g))</td>
<td>0.7350</td>
<td>0.8788</td>
<td>0.8199</td>
<td>0.7314</td>
<td>0.7106</td>
<td>0.6967</td>
<td>0.7012</td>
</tr>
<tr>
<td>LCG(3249286849523012805,1,2(^{63}))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\mu_t(m,g))</td>
<td>2.4594</td>
<td>2.4281</td>
<td>3.7081</td>
<td>2.8333</td>
<td>3.7633</td>
<td>3.0844</td>
<td>1.9471</td>
</tr>
<tr>
<td>(S_t(m,g))</td>
<td>0.8234</td>
<td>0.7428</td>
<td>0.7829</td>
<td>0.7176</td>
<td>0.7350</td>
<td>0.6991</td>
<td>0.6451</td>
</tr>
</tbody>
</table>